Athens Journal of Technology and Engineering - Volume 8, Issue 2, June 2021 – Pages 169-180

Numerical Uncertainty in Parallel Processing Using Computational Fluid Dynamics as Example

By Mark Lin[*] & Periklis Papadopoulos[±]

Computational methods such as Computational Fluid Dynamics (CFD) traditionally yield a single output – a single number that is much like the result one would get if one were to perform a theoretical hand calculation. However, this paper will show that computation methods have inherent uncertainty which can also be reported statistically. In numerical computation, because many factors affect the data collected, the data can be quoted in terms of standard deviations (error bars) along with a mean value to make data comparison meaningful. In cases where two data sets are obscured by uncertainty, the two data sets are said to be indistinguishable. A sample CFD problem pertaining to external aerodynamics is copied and ran on 29 identical computers in a university computer lab. The expectation is that all 29 runs should return exactly the same result; unfortunately, in a few cases the result turns out to be different. This is attributed to the parallelization scheme which partitions the mesh to run in parallel on multiple cores of the computer. The distribution of the computational load is hardware-driven depending on the available resource of each computer at the time. Things, such as load-balancing among multiple Central Processing Unit (CPU) cores using Message Passing Interface (MPI) are transparent to the user. Software algorithm such as METIS or JOSTLE is used to automatically divide up the load between different processors. As such, the user has no control over the outcome of the CFD calculation even when the same problem is computed. Because of this, numerical uncertainty arises from parallel (multicore) computing. One way to resolve this issue is to compute problems using a single core, without mesh repartitioning. However, as this paper demonstrates even this is not straight forward.

Keywords: *numerical uncertainty, parallelization, load-balancing, automotive aerodynamics*

Introduction

If you will, imagine punching in 2 plus 2 on a calculator a million times and expect a different answer than 4 each time. Taking it one step further, what if a software program is ran over and over on a computer? Lastly, what if this process is repeated for many days, weeks even? Would one expect a different result? The answer should be no, right? This is important because it describes the basic operation of a numerical solver, for example CFD, to arrive at a single solution (Eça and Hoekstra 2014).

[*]Graduate Student, San Jose State University, USA.
[±]Professor, San Jose State University, USA.

https://doi.org/10.30958/ajte.8-2-3 doi=10.30958/ajte.8-2-3

The motivation for this study is that previously, when one computer is not available in the lab (for example, another student is logged on), the author would get on another computer to run his CFD code. Overtime, he has observed that even when the same code is running the result would not replicate sometimes (Beckwith et al. 1993). This troubles him. Before we start let's be mindful that the computers are identical, and likely purchased in a batch order from the same manufacturer Dell.

The Problem

The situation that is being described here is exactly how a CFD code functions. It is a computer program with many lines of code, run over-and-over again, for many days. A CFD code computes the solution to five coupled, partial differential equations (the Navier-Stokes Equations). While solving a partial differential equation on the computer is a complicated task according to Hennig et al. (2015), the exact lines of computer instruction sent to the hardware should be the same even when the physical hardware is different (although with the same specifications).

This paper addresses the result inconsistency across a set of identical computers. The precise problem that is investigated in this paper is to take the same CFD code, copy it onto a bunch of identical computers (Figure 1), run it for exactly the same amount of time (i.e., iterations) and report the finding. If the result turns out to be different, comment on how to proceed in that situation.

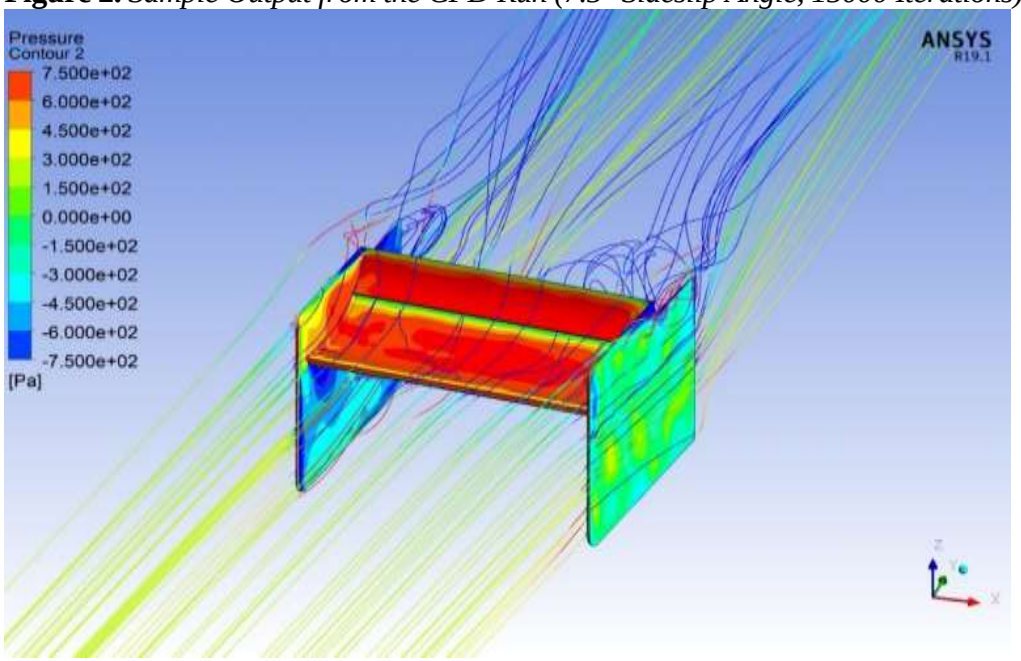
Figure 1. *The Group of Computers that are Used in this Study*

One assumption in this study is that because the CFD software is a commercial software (ANSYS Fluent), due to its proprietary nature ANSYS Technical Support cannot comment on the repeatability of the source code. They can only provide consultation and it is up to the authors to research the root cause. Furthermore,

numerical uncertainty as it relates to computation problems encompasses a wide body of research into areas such as mathematical formulation of a physical problem (Cruse and Rizzo 1968), numerical solution methods (Press et al. 2007), and computer programming and processors (Landau et al. 2008). While CFD is used as an example in this study, the problem this paper examines has to do specifically with computer programming and processors.

The case that's to be used to illustrate is a simple fluent simulation from a study presented at the 2nd ATINER Mechanical Engineering Conference (Lin and Papadopoulos 2018). The simulation is a rear wing assembly from a racecar model in angled flow (yaw). An example of the output after four days of computation is shown in Figure 2. The metrics that are used for comparison are the resultant downforce and the resultant drag force from ANSYS's CFD-Post module.

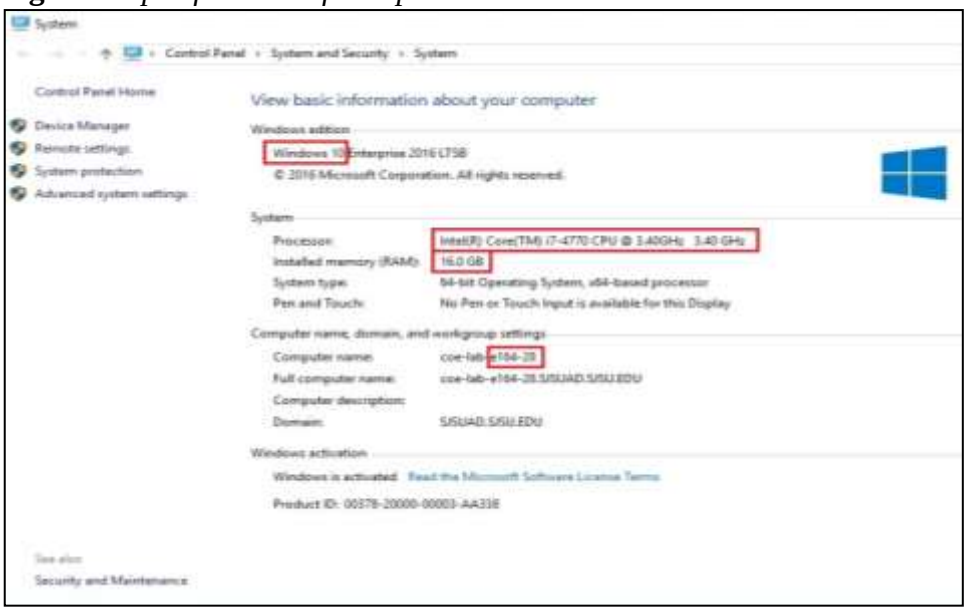
Figure 2. *Sample Output from the CFD Run (7.5° Sideslip Angle, 15000 Iterations)*



Approach

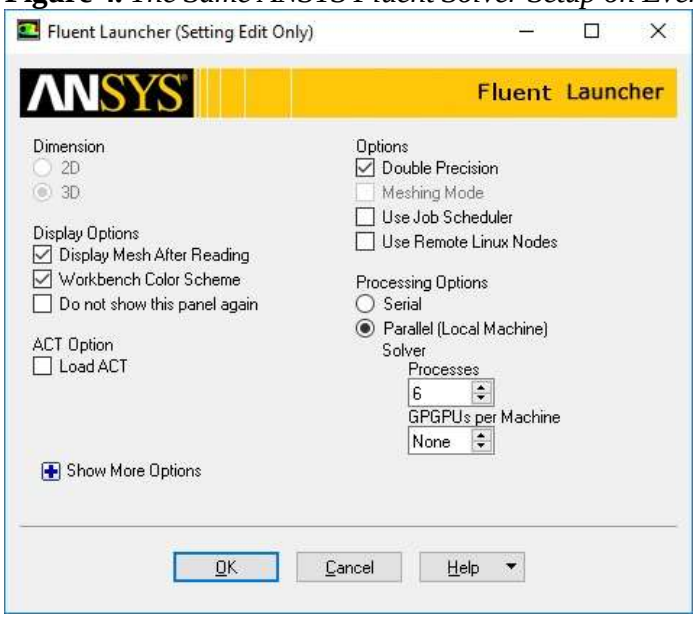
In this study, 29 computers at San Jose State University's Engineering 164 computer lab (a.k.a. ENG164) are used. Every computer has exactly the same specifications as shown in Figure 3. Each computer has an Intel Core i7 processor with 8 cores (i.e., 4 physical cores and 4 virtual cores) that can be used for parallel computing (López Azaña 2015). The computers also have 16 GB of RAM and 1TB of hard drive. Among the 29 computers, there is one exception: computer no. E164-30. While this computer looks the same as the others, it turns out that it has a different processor: an Intel Core i5 processor with only 4 cores. However, as the result shows it behaves the same as the other 28 computers.

Figure 3. *Specifications of Computers in ENG164 Lab*



The CFD program in this study is a commercial software ANSYS Fluent (release 19.1 to 2019 R3) that is installed on every computer (ANSYS Manual 2014). Each copy of Fluent solver is ran in double-precision, parallel processing, utilizing 6 out of 8 cores as shown in Figure 4. For the exception case E164-30, only 3 cores were used. From this ANSYS Fluent solver description, it is argued that all software is identical on every computer to calculate the CFD problem. On average, each computer takes about 4 days to finish running 15000 iterations. In Fluent the residual checking option is turned off so that every computer can run to exactly 15000 iterations without terminating even if the convergence criteria is met earlier.

Figure 4. *The Same ANSYS Fluent Solver Setup on Every Computer*



Within ANSYS Fluent, the case that is setup to run is a simple two-airfoil rear wing from a racecar model. The geometry is simple, and it is rotated 7.5° with respect to the enclosure. The freestream is also rotated 7.5° so that it is coming in head-on to the leading edge of the airfoils. The rear wing assembly is placed in the mid-height of the enclosure so there is no ground effect, and the only boundary layer is on the wing itself. This set of boundary conditions is chosen because we want to simulate as many non-symmetric 3D boundary conditions as possible. The solver that is used is the Transition SST turbulence model, with a CFL number of 0.5. This is a convenient simulation to run because it has been performed previously. The same code is simply copied from an external hard drive onto every computer so no further changes to the code are made to run on each computer.

Multicore (Parallel) Computation Results

As shown in Table 1, 25 out of 29 results are identical but 4 of them are different. Initially this result causes much headache. The computers are identical running the same code so there really is no reason for this behavior. While some may argue that the difference is small, less than 1%, but the important question is why are not they exactly the same?

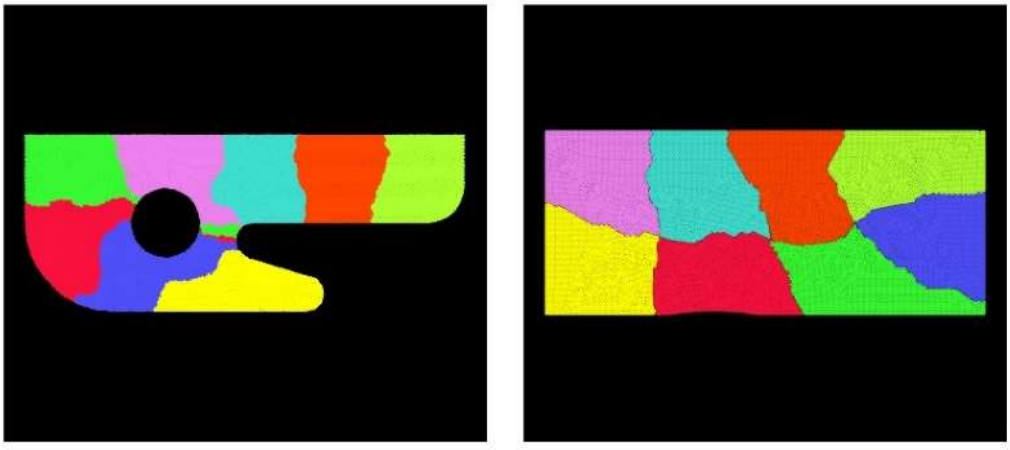
Table 1. *Summary of Multicore Results from 29 Computers in ENG164 Computer Lab*

Computer	Downforce (Fz)	Drag Force (Fy)	Drag Force (Fx)	Combined Drag Force
E164-01	-560.705	198.408	35.0799	201.4853
E164-02	-560.705	198.408	35.0799	201.4853
E164-03	-560.705	198.408	35.0799	201.4853
E164-04	-560.705	198.408	35.0799	201.4853
E164-05	-560.705	198.408	35.0799	201.4853
E164-06	-560.705	198.408	35.0799	201.4853
E164-07	-560.705	198.408	35.0799	201.4853
E164-08	-560.705	198.408	35.0799	201.4853
E164-09	-560.705	198.408	35.0799	201.4853
E164-10	-560.705	198.408	35.0799	201.4853
E164-11	-560.705	198.408	35.0799	201.4853
E164-12	-560.705	198.408	35.0799	201.4853
E164-13	-560.705	198.408	35.0799	201.4853
E164-14	-560.705	198.408	35.0799	201.4853
E164-15	-560.705	198.408	35.0799	201.4853
E164-16	-560.705	198.408	35.0799	201.4853
E164-17	-560.706	198.408	35.0808	201.4855
E164-18	-560.705	198.408	35.0799	201.4853
E164-19	-560.705	198.408	35.0799	201.4853
E164-20	-560.705	198.408	35.0799	201.4853
E164-21	-560.705	198.408	35.0799	201.4853

E164-22	-560.19	198.714	34.9482	201.7638
E164-23	-560.705	198.408	35.0799	201.4853
E164-24	-560.705	198.408	35.0799	201.4853
E164-25	Computer removed from **ENG164**			
E164-26	-560.705	198.408	35.0799	201.4853
E164-27	-560.705	198.408	35.0799	201.4853
E164-28	-560.705	198.408	35.0799	201.4853
E164-29	-560.706	198.409	35.0796	201.4863
E164-30	-560.97	198.941	35.249	202.0396

As it turns out, the use of parallel computing with multiple CPU cores running simultaneously has an effect on the consistency of result. Walshaw and Cross (2007) explained that when in parallel computing the mesh is partitioned into different blocks. Each block is sent to a different core (CPU) and the load between each block is distributed and balanced. The CPU load on each core is distributed differently depending on the available RAM size, the hard drive space, and the other Windows processes that are running. This part of the computing process is done automatically and is entirely transparent to the user so we have no control over how the blocks are divided up. Software algorithm such as JOSTLE or METIS (this is used by ANSYS) is used to perform this task. An illustration of this redistribution process is shown in Figure 5. This parallelization difference is further confirmed by Mishra and Aggarwal in 2011, in their paper "ParTool: A Feedback-Directed Parallelizer", presented at the 9th International Symposium on Advanced Parallel Processing Technologies (Mishra and Aggarwal 2001). This fact has been confirmed by ANSYS Application Support that parallel computing would most likely not lead to the same result (Kishore A, personal communication, July 29, 2019 to May 28, 2020).

Figure 5. *Example Graphs that are Mapped onto Two Different Processor Configurations*



(a) mapping onto a 1D array (b) mapping onto a 2D array

Source: Walshaw and Cross 2007.

Single-Core Computation Results

While ongoing discussion has been focused on parallel computing utilizing the multicores of an Intel Core™ i7-4770 processor, it has been explained to us why when using parallel computing, the result may be different. Next, the study is switched to utilizing single-core processing to make the comparison fair (Maity et al. 2013). This is not an easy decision to make because the code will take longer to run in the single-core mode. The processing time is more than 10 days to complete. In addition, we have to ensure the runs will not be interrupted during the 10-day period, either by students or by Windows update. The single-core results are shown in Table 2.

Table 2. *Summary of Single-Core Results from 29 Computers in ENG164 Computer Lab*

Computer	Downforce (Fz)	Drag Force (Fy)	Drag Force (Fx)	Combined Drag Force
E164-01	-559.807	199.02	44.4015	203.915
E164-02	-559.807	199.02	44.4015	203.915
E164-03	-559.807	199.02	44.4015	203.915
E164-04	-559.807	199.02	44.4015	203.915
E164-05	-559.807	199.02	44.4015	203.915
E164-06	-559.807	199.02	44.4015	203.915
E164-07	-559.807	199.02	44.4015	203.915
E164-08	-559.807	199.02	44.4015	203.915
E164-09	-559.807	199.02	44.4015	203.915
E164-10	-559.807	199.02	44.4015	203.915
E164-11	-559.807	199.02	44.4015	203.915
E164-12	-559.807	199.02	44.4015	203.915
E164-13	-559.807	199.02	44.4015	203.915
E164-14	-559.807	199.02	44.4015	203.915
E164-15	-559.807	199.02	44.4015	203.915
E164-16	-559.807	199.02	44.4015	203.915
E164-17	-559.807	199.02	44.4015	203.915
E164-18	-559.807	199.02	44.4015	203.915
E164-19	-559.807	199.02	44.4015	203.915
E164-20	-559.807	199.02	44.4015	203.915
E164-21	-559.807	199.02	44.4015	203.915
E164-22	-559.807	199.02	44.4015	203.915
E164-23	-559.807	199.02	44.4015	203.915
E164-24	-559.807	199.02	44.4015	203.915
E164-25	Computer removed from **ENG164**			
E164-26	-559.807	199.02	44.4015	203.915
E164-27	-559.807	199.02	44.4015	203.915
E164-28	-559.807	199.02	44.4015	203.915
E164-29	-559.807	199.02	44.4015	203.915
E164-30	-559.807	199.02	44.4015	203.915

A careful note must be said about how this single-core result is achieved: it is through much trial and tribulation. Therefore one should not expect, and normally

would not achieve, consistent result such as this even if one were to run in single-core mode without the care and control that is exercised in this academic study.

From this result there are three important things to note: first is that all the results are identical, which is what we set out to prove. However, in order to get to this stage there are many details unsaid in this statement. After numerous 10-day runs on 29 computers, we have learned that it is important to look at the residuals. If the residuals are not the same at the end of the run (at 15000 iterations), the integral results of downforce and drag force would not be the same. Secondly, if the solving process starts off at a different starting point, meaning with a different set of initial conditions, no matter how close they are to begin with, the result will never be the same no matter how long the case runs (Alayil 2016). A sample of the initial value is shown in Figure 6. This taught us that the current problem is not so much about convergence as it is about number crunching. Once the residual values start to deviate as the iteration progresses, the difference would start to propagate from one residual value to the next. One note about the initialization scheme is that there are two types available in ANSYS: hybrid initialization and standard initialization. For this study hybrid initialization is used. Hybrid initialization requires some guessing on Fluent's part by solving the Euler equation on a coarse mesh and use that as the initial guess, as oppose to standard initialization where the user has to specify all the initial parameters which leads to a longer solve time.

In ANSYS the residual is written as

$$a_p \phi_p = \sum_{nb} a_{nb} \phi_{nb} + b \tag{1}$$

$$R^{\phi} = \| \sum_{nb} a_{nb} \phi_{nb} + b - a_p \phi_p \| \tag{2}$$

a_p is the center coefficient, a_{nb} is the influence coefficient for the neighboring cell, and b is the contribution of the source term. The residual is the higher order Taylor Series terms that are left out when the conservation equation is discretized. This discretization error is tracked iteration-to-iteration, and it's the change in residuals that's plotted. Typically, we want the change in residuals to be smaller than three orders of magnitude from the start. Therefore, to have all 29 computers produce the same result, one needs to first make sure that their residuals are the same. The trick to solve this problem is to check the residuals during the 15000 iterations to see when they start to deviate.

Figure 6. *Sample Initialization and Residuals from Fluent Solver*

```
/***Include Solver Example***/
-Case will be initialized with constant pressure
   iter    scalar-0
   1    1.000000e+00
   2    1.463416e-03
   3    2.631864e-04
   4    8.668557e-05
   5    2.347466e-05
   6    7.955006e-06
   7    2.599028e-06
   8    9.270428e-07
   9    3.401770e-07
   10   1.448375e-07
Hybrid initialization is done.
  iter  continuity     x-velocity     y-velocity     z-velocity     energy         k              omega          intermit
        retheta        cl-1           cd-1           surf-mon-2     surf-mon-1     time/iter
 Reversed flow on 3 faces (2.4% area) of pressure-outlet 6.
    1   1.0000e+00     1.0000e+00     1.0000e+00     1.0000e+00     1.0000e+00     8.0242e-01     4.4196e-01     5.4988e-04
        1.1375e+02     1.2150e+01     1.5784e+00     1.1985e+01     -3.8831e+02    345:48:37      14999
    2   8.3024e-01     8.7096e-01     8.7202e-01     7.4272e-01     8.2991e-01     6.7417e-01     4.1124e-01     3.2043e-03
        9.5692e-01     1.5757e+01     1.2060e+01     1.1981e+01     -3.6751e+02    372:27:01      14998
    .
    .
    .
14999   6.8973e-03     2.9161e-04     2.0077e-03     3.1864e-03     6.8967e-03     6.0816e-04     1.4815e-04     4.5492e-07
        1.5752e-05     3.2611e+02     7.2449e+01     1.4839e+01     1.0764e+02     0:01:05        1
15000   6.8688e-03     2.9147e-04     2.0052e-03     3.1852e-03     6.8681e-03     6.0817e-04     1.5489e-04     4.5793e-07
        1.5808e-05     3.2614e+02     7.2297e+01     1.4839e+01     1.0761e+02     0:00:00        0
   Writing "| gzip -2cf > SolutionMonitor.gz"...
   Writing temporary file C:\\Users\\010815~1\\AppData\\Local\\Temp\\fintgz-152043002 ...
Done.
/****End Solver Example***/
```

The third important thing is that during the solving process, the run cannot be interrupted and restarted, otherwise the residuals would also start to deviate. This is attributed to a coding error in the interrupt/resume routine in Fluent as shown in Figure 7. Note that the last four residual values that are highlighted in yellow: once the program is stopped and restarted again, these four values would flip in order. Prior to realizing this, it has been a common practice for people to pause the solution, examine the result, and restart again. However as we've shown here, doing this would distort the residuals field. Result will never be the same again so all runs have to be completed in one-shot without stopping.

Obeying these do-not-do rules, what works eventually is to initialize the program just once for all 29 computers, run to completion without stopping, examine the last iteration residuals, and plot the integral results for all 29 computers. As previously noted, there are too many details in a commercial CFD program that could cause the results to differ, which has been faithfully demonstrated in this study. As this study shows, the result does not have to be different - one just needs to be extremely careful to eliminate all the possibilities.

Figure 7. *Example of the Solver Process that is Interrupted and then Restarted Again*

```
/***Example of Solver Interruption***/
  iter  continuity     x-velocity     y-velocity     z-velocity     energy         k              omega          intermit
        retheta        cl-1           cd-1           surf-mon-2     surf-mon-1     time/iter
    17  1.2174e-01     2.7499e-02     6.1869e-02     1.0322e-01     1.2131e-01     2.8837e-02     4.9984e-02     4.0734e-03
        1.0671e-02     5.9466e+01     1.0792e+02     1.1864e+01     -3.3634e+02    282:45:05      14983
    18  1.2337e-01     2.6605e-02     5.5318e-02     9.1153e-02     1.2291e-01     2.6485e-02     4.5585e-02     3.9282e-03
        9.5939e-03     6.1226e+01     1.1132e+02     1.1851e+01     -3.3802e+02    280:17:15      14982
 Reversed flow on 1 face (0.5% area) of pressure-outlet 6.
    19  1.2550e-01     2.5063e-02     5.2885e-02     8.7777e-02     1.2505e-01     2.4458e-02     4.1788e-02     3.7698e-03
        8.7626e-03     6.2778e+01     1.1437e+02     1.1837e+01     -3.4098e+02    279:08:44      14981
Writing "| gzip -2cf > FFF-30.1-1.cas.gz"...
Writing temporary file C:\\Users\\010815~1\\AppData\\Local\\Temp\\flntgz-300645 ...
/***Interrupt Solver***/

/***Restart Solver***/
  iter  continuity     x-velocity     y-velocity     z-velocity     energy         k              omega          intermit
        retheta        cl-1           cd-1           surf-mon-2     surf-mon-1     time/iter
    19  1.2550e-01     2.5063e-02     5.2885e-02     8.7777e-02     1.2505e-01     2.4458e-02     4.1788e-02     3.7698e-03
        8.7626e-03     -3.4098e+02    1.1837e+01     1.1437e+02     6.2778e+01     0:00:00        14981
 Reversed flow on 1 face (0.5% area) of pressure-outlet 6.
Writing "| gzip -2cf > status.txt.gz"...
Writing temporary file C:\\Users\\010815~1\\AppData\\Local\\Temp\\flntgz-268602 ...
Done.
    20  1.2639e-01     2.3437e-02     5.0661e-02     8.4820e-02     1.2595e-01     2.2697e-02     3.8489e-02     3.6125e-03
        8.0446e-03     -3.4385e+02    1.1823e+01     1.1711e+02     6.4131e+01     349:32:00      14980
 Reversed flow on 1 face (0.5% area) of pressure-outlet 6.
    21  1.2619e-01     2.1551e-02     4.8649e-02     8.2233e-02     1.2579e-01     2.1150e-02     3.5593e-02     3.4831e-03
        7.4447e-03     -3.4935e+02    1.1809e+01     1.1954e+02     6.5273e+01     345:20:57      14979
/****End Interrupt Example***/
```

Discussion

Now that data has been shown and conclusion drawn that not all CFD runs produce exactly the same result (unless extreme caution is exercised), it is time to offer an alternative way of looking at CFD data, namely to see it as a statistical distribution rather than as a precise number. In statistics, the result is typically quoted as an average with an error bar (Huff 1993). This makes comparison of two distributions possible by saying whether one is better than the other when it's outside of data uncertainty; or conversely, by saying whether two datasets are indistinguishable when the mean value is overlapping within the error bars. In the multicore computation case here, the mean value (μ) from 29 runs is 560.696 N and the standard deviation (σ) is 0.109 N. Hence, if a normal distribution is assumed then the result can be quoted as $\mu \pm 3\sigma$ to encompass 99.7% of the population (Devore 1991). For this dataset (n=29), the downforce would be 560.696 ± 0.327 N and the drag force would be 201.514 ± 0.342 N.

When a computer is utilizing multiple cores, it is performing parallel computing: the program is broken into smaller blocks, sent to different processors, and all are running at the same time. In that case the instructions are not executed sequentially but in parallel, and the intermediate results are passed back-and-forth between the different CPU's. Parallelization uses Message Passing Interface (MPI) to pass information in a distributed memory environment which is the de-facto standard in HPC systems (Fagg et al. 2001). Fluent offers three MPI choices: PCMPI, Microsoft MPI, and Intel MPI which adds another variable to data consistency (Sharcnet 2019). In practical computing, parallel processing utilizing

multiple cores is still the norm and offers significant advantage in shorter solve time (Ali and Khan 2012). Therefore, acknowledging that the result will be different due to these inherent uncertainties, the use of statistical averaging and data distribution is warranted (Oliver et al. 2014).

Conclusions

This study has taken two years to complete because of the long solve time and the availability of 29 identical computers in a university engineering lab setting. The data shows that while most of the time the computational result is exactly the same, but occasionally the result would differ. We are happy to report we can finally achieve the same result on all 29 computers. This time by having access to 29 identical computers in the ENG164 computer lab during COVID-19 lockdown, a definitive view of how computational results could vary has been demonstrated.

It has been shown that the same CFD code, running on identical computers, and using the same solver, while most of the time would yield the same result, but once-in-a-while the result would not repeat. This is the main conclusion from this paper. Since it has been demonstrated that this can happen, the idea of quantifying a statistical error is not unreasonable. It is now possible to calculate an average and a standard deviation for computational results. Therefore, when reporting computational results from a parametric study where a design parameter is varied, in addition to doing a curve fit through the data points one should also show error bars on the data set. This would make comparison of different designs more meaningful and allow real trends to be seen outside of numerical uncertainty.

Acknowledgments

The authors would like to thank Prof. N. Mourtos of San Jose State University for granting access to ENG164 lab that provides the 29 computers that are used in this study. The authors would also like to thank ANSYS Corporation for providing the Fluent license that is used in this research, and the consultation with ANSYS technical support staff.

References

Alayil R (2016) *Why does the CFD Results change with different initialization techniques?* ResearchGate Discussion Board.

Ali J, Khan RZ (2012) Performance analysis of matrix multiplication algorithms using MPI. *International Journal of Computer Science and Information Technologies* 3(1): 3103–3106.

ANSYS Inc. (2014) *Introduction to ANSYS fluent, 15.0 release*. Training Manual, 1st Edition, Inventory #000575.

Beckwith T, Marangoni R, Lienhard J (1993) *Mechanical measurements*. 5th Edition. Reading, Massachusetts: Addison-Wesley Publishing Company.

Cruse TA, Rizzo FJ (1968) A direct formulation and numerical solution of the general transient elastodynamic problem. *Journal of Mathematical Analysis and Applications* 22(1): 244–259.

Devore J (1991) *Probability and statistics for engineering and the sciences*. 8th Edition. Boston, Massachusetts: Brooks/Cole Publishing Company.

Eça L, Hoekstra M (2014) A procedure for the estimation of the numerical uncertainty of cfd calculations based on grid refinement studies. *Journal of Computational Physics* 262C(Apr): 104–130.

Fagg GE, Bukovsky A, Dongarra J (2001) HARNESS and fault tolerant MPI. *Parallel Computing* 27(11): 1479–1495.

Hennig P, Osborne MA, Girolami M (2015) Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A* 471(2179).

Huff D (1993) *How to lie with statistics*. New York: W. W. Norton & Company, Inc.

Landau RH, Paez MJ, Bordeianu CC (2008) *A survey of computational physics. Chapter 14 high-performance and parallel computing hardware, and programming for it*, 352–389. Princeton, New Jersey: Princeton University Press.

Lin M, Papadopoulos P (2018) Application of computer aided design tools in CFD for computational geometry preparation. In *ATINER Conference Paper Proceedings Series* (Athens, Greece, 24–28 July 2018).

López Azaña D (2015) *Differences between physical CPU vs logical CPU vs Core vs Thread vs Socket*. Retrieved from: https://www.daniloaz.com/en/differences-between-physical-cpu-vs-logical-cpu-vs-core-vs-thread-vs-socket/. [Accessed 6 October 2020]

Maity S, Bonthu SR, Sasmal K, Warrior H (2013) Role of parallel computing in numerical weather forecasting models. In *IJCA Special Issue on International Conference on Computing, Communication and Sensor Network* 4(Mar): 22–27.

Mishra V, Aggarwal SK (2011) ParTool: a feedback-directed parallelizer. *Advanced Parallel Processing Technologies* 6965(2011): 157–171.

Oliver TA, Malaya N, Ulerich R, Moser RD (2014) Estimating uncertainties in statistics computed from direct numerical simulation. *Physics of Fluids* 26(3).

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007) *Numerical recipes: the art of scientific computing*. 3rd Edition. New York: Cambridge University Press.

Sharcnet (2019) *Starting parallel ANSYS fluent on a windows system using command line options*. Sharcnet.

Walshaw C, Cross M (2007) JOSTLE: parallel multilevel graph-partitioning software - An overview. *Mesh Partitioning Techniques and Domain Decomposition Techniques*, 10.4203/csets.17.2.